ЗАПИС (СТРУКТУРА)

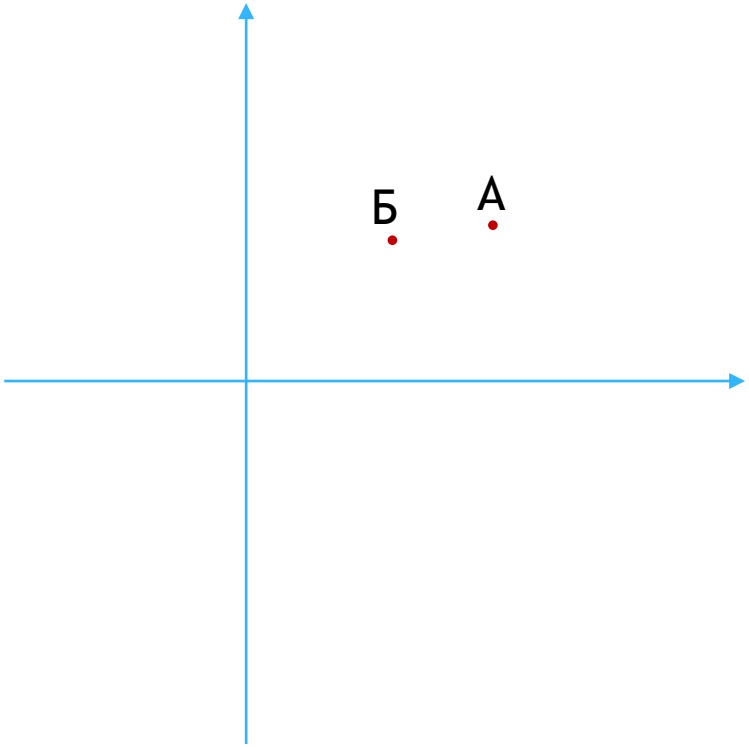
доц. д-р Нора Ангелова

ЗАПИС (СТРУКТУРА)

- **Идея**
  Да създадем комплексен „тип“ от данни с множество от стойности от различни типове.
  Да обединим множество стойности, които могат да бъдат от различен тип, в една структура и да създаваме различни нейни инстанции.

Запис (структура)

- Нека използваме структура, описваща точка в равнината



Запис (структура)

- Полета(член-данни) на структурата – елементите, които характеризират/описват структурата

Запис (структура)

- Декларация на структури

```
struct <име_на_структура>;
```

- Дефиниция на структура

```
struct [<име_на_структура>]опц {
        <дефиниция_на_полета>;
        {<дефиниция_на_полета>;}опц
}[<име_на_обект>]опц;
```

```
<дефиниция_на_полета> ::= <тип> <име_поле> {,<име_поле>}опц;
<име_на_структура>, <име_поле> ::= <идентификатор>
<име_на_обект> ::= <идентификатор>
<тип> ::= <име_на_тип>|<дефиниция_на_тип>
```

Забележка: Обикновено имената на структурите следва PascalCase конвенцията (първата буква на всяка дума е главна, включително и първата)

Може да се създаде анонимна структура – името да бъде пропуснато. Тогава трябва да се създаде инстанция на тази структура между ‘}’ и знака ‘;’

Запис (структура)

- Област на декларирана/дефинирана структура
  - за всички функции след декларация/дефиницията на структурата
  - в рамките на функцията
  - в рамките на блока

Запис (структура)

- Множество от стойности

Пример:

```
// Пример 1 - обща структура
struct ExampleStructName {
  int field1;
  double field2;
};
```

Всички двойки от вида:
{int, double}

```
// Пример 2 - Точка в равнината
struct Point2D {
  double x;
  double y;
};
```

Всички двойки от вида:
{double, double}

Запис (структура)

- Създаване на конкретни инстанции от тип структура – обекти

<обект> ::=
<име_на_структура> <идентификатор> [={<редица_от_изрази>}]$_{опц}$
{, <идентификатор> [={<редица_от_изрази>}]$_{опц}$}$_{опц}$

Примери:

```
ExampleStructName obj1;
ExampleStructName obj2, obj3 = {5, 1.5};
```

Запис (структура)

- Достъп до полетата на структурата
  - достъпът до полетата на структура е пряк.
  - осъществява се чрез обект.
  - обектът и името на полето се разделят с оператора точка.
    <обект>.<име_на_поле>

Пример:

```
// Пример 1 - обща структура
struct ExampleStructName {
  int field1;
  double field2;
};
ExampleStructName obj = {5, 1.5};
obj.field1;

// Пример 2 - Точка в равнината
struct Point2D {
  double x;
  double y;
};

Point2D p1 = {1.3, 2.4};
p1.x; p1.y;
```

Запис (структура)

- **Памет**

Дефиницията на запис НЕ предизвиква отделянето на памет за съхраняване на полетата ѝ.

Памет се заделя при създаването на обект от тип запис (структура).

ЗАПИС (СТРУКТУРА)

- **Представяне в паметта**

Заделя се памет за всяко поле на променливата

Пример:

```
struct Example {
   char a;
   short int b;
   int c;
   char d;
};
```

Очакваме заделената памет за структурата да изглежда по следния начин:

Size of 1 block = 1 byte

Size of 1 row = 4 byte

a	b	b	c
c	c	c	d

ЗАПИС (СТРУКТУРА)

- **Представяне в паметта**

Процесорът не може да достъпи паметта – използва се равен размер на думите (4B - max size на полето).

Всяко поле трябва да се разположи в паметта на адрес, който е равен на число кратно на размера на полето.

Пример:

```
struct Example {
  char a;
  short int b;
  int c;
  char d;
};
```

Реално представяне

Size of 1 block = 1 byte

Size of 1 row = 4 byte

a	padding	b	b
c	c	c	c
d	padding	padding	padding

ЗАПИС (СТРУКТУРА)

- **Представяне в паметта**

```
struct Example {
  char a;
  short int b;
  int c;
  char d;
};

Example myFirstExampleObj;
sizeof(myFirstExampleObj) != sizeof(char) +
       sizeof(short int) + sizeof(int) + sizeof(char)
```

Size of 1 block = 1 byte

Size of 1 row = 4 byte

a	padding	b	b
c	c	c	c
d	padding	padding	padding

Запис (структура)

- **Представяне в паметта**

```
struct Test1 {
  short s; /* 2 bytes */ + /* 2 padding bytes */
  int i;   /* 4 bytes */
  char c;  /* 1 byte  */ + /* 3 padding bytes */
};

struct Test2 {
  int i;   /* 4 bytes */
  short s; /* 2 bytes */
  char c;  /* 1 byte  */ + /* 1 padding byte */
};

const int sizeTest1 = sizeof(struct Test1); /* = 12 */
const int sizeTest2 = sizeof(struct Test2); /* = 8 */
```

Запис (структура)

От направения анализ могат да се направят следните заключения:

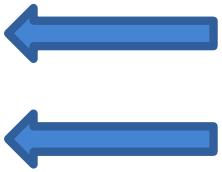
- Полетата се сортират по тяхната големина в низходящ ред.
- Отстъпите (paddings) са позволени между данните за отделните полета и след последното поле.

Запис (структура)

- Имената на полетата в рамките на една дефиниция на структурата трябва да са различни идентификатори.

Пример:

```
struct ExampleStructName {
  int field1;
  double field2;
};
```



ЗАПИС (СТРУКТУРА)

- Възможно е име на структура, на нейно поле и на произволна променлива в програмата да е един и същ идентификатор.

X **НЕ ГО ИЗПОЛЗВАЙТЕ!**

Запис (структура)

- Възможно е влагане на структури, т.е.
  поле на структура може да е от тип структура.

Пример:

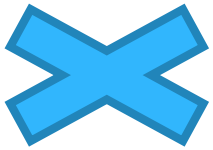
```
struct Student {
  //...
};

struct ClassRoom {
  Student student1;
};
```

Запис (структура)

- Не е възможно поле на структурата да е от тип, съвпадащ с името на структурата.

Пример:

```
struct Student {
  Student student1;
};
```



* *Какъв би бил размерът на Student?*

Запис (структура)

- Възможно е поле на структурата да е от тип указател към името на структурата.

Пример:

```
struct Student {
  Student * student1;
};
* Какъв е размерът на Student?
```

Запис (структура)

- Две структури, които се обръщат една към друга

Пример:

```
struct StudentList; // Декларация на втората
struct Student {
  // ...
  StudentList * friends;
};

struct StudentList {
  // ...
  Student st;
};
```

Запис (структура)

- **Дефиниция на указател към запис**

<указател_към_структура> ::=

<име_на_структура>* <променлива_указател>
[=&{<променлива>}]$_{опц}$;

<променлива> ::= <име_на_структура>

Примери:

```
ExampleStructName obj1;
ExampleStructName * objPtr = &obj1;
```

Запис (структура)

- **Достъп до полетата на структурата чрез указател към структура**

Примери:

```
ExampleStructName obj1;
ExampleStructName* objPtr = &obj1;


(*objPtr).field1; // Достъп до field1
objPtr->field1;   // Достъп до field1
```

Запис (структура)

- **Масиви**

Примери

```
Point2D points[10];
points[0];
std::cout << points[0].x << “ ” << points[0].y;

for(int i=0; i<10; i++) {
    printPoint(points[i]);
}
```

Запис (структура)

- Как да работим с член-данните
  - Добавяне на валидация
  - Възможност за директно отпечатване на цялата информация
  - др.

Член-функции(методи)

- Член-функциите са функции, които работят с член-данните на обекта от дадена структура.

ЗАПИС (СТРУКТУРА)

- **Декларация на член-функция**

```
struct [<име_на_структура>]опц {
        <дефиниция_на_полета>;
        {<дефиниция_на_полета>;}опц
        {<декларация_член_функции>;|{< декларация_член_функции >;}опц}опц
}[<променлива>]опц;
```

```
<дефиниция_на_полета> ::= <тип> <име_поле> {,<име_поле>}опц
<име_на_структура>, <име_поле> ::= <идентификатор>
<променлива> ::= <идентификатор>
<тип> ::= <име_на_тип>|<дефиниция_на_тип>
<декларация_член_функции> ::=
 <тип> <име> ({<списък_формални_параметри>}опц);
```

Забележка: Обикновено имената на структурите следва PascalCase конвенцията (първата буква на всяка дума е главна, включително и първата)_

Може да се създаде анонимна структура – името да бъде пропуснато. Тогава трябва да се създаде инстанция на тази структура между ‘}’ и знака ‘;’

Запис (структура)

- За примера ще използваме нашата точка в равнината

```
// Точка в равнината
struct Point2D {
  double x;
  double y;
};
```

Запис (структура)

- **Декларация на член-функции**

Пример:

```
struct Point2D {
  double x;
  double y;
  void print(); // Функция, която извежда координатите на точка в
                // равнината
};
```

КЛАСОВЕ

- Дефиниция на член-функции, които са декларирани в структурата (външна дефиниция)

```
struct [<име_на_структура>]опц {
        <дефиниция_на_полета>;
        {<дефиниция_на_полета>;}опц
        <тип> <име> ({<списък_формални_параметри>}опц);
}[<променлива>]опц;

// Дефиниция извън тялото на структурата
<дефиниция_член_функции> ::=
<тип> <име_на_структура>::<име>({<списък_формални_параметри>}опц) {
  <тяло>
}
```

КЛАСОВЕ

- Дефиниция на член-функция на структура

Пример:

```
struct Point2D {
  double x;
  double y;
  void print();
};


void Point2D::print() {
  std::cout << x << " " << y;
  // OR std::cout << '(' << x << ',' << y << ')';
}
```

Запис (структура)

- Вградени(inline) член-функции – дефиницията е в тялото на класа.
  - Тялото на функцията се замества на мястото на извикването

Пример:

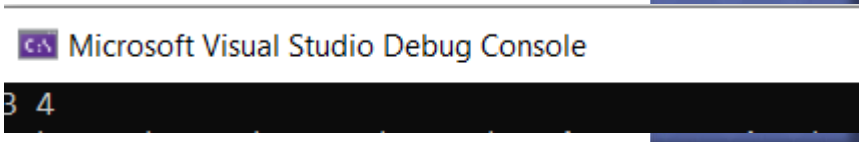
```
struct Point2D {
  double x;
  double y;
  void print() {
    std::cout << x << " " << y;
    // OR std::cout << '(' << x << ',' << y << ')‘;
  }
};
```

ЗАПИС (СТРУКТУРА)

- Извикване на член-функции
  - Работят с член-данните на класа
  - Извикват се с обект на класа

Пример:

```
struct Point2D {
  double x;
  double y;
  void print();
};


void Point2D::print() {
  std::cout << x << " " << y;
  // OR std::cout << '(' << x << ',' << y << ')';
}

int main() {
    Point2D myFirstPoint = { 3, 4 };
    myFirstPoint.print();
    return 0;
}
```

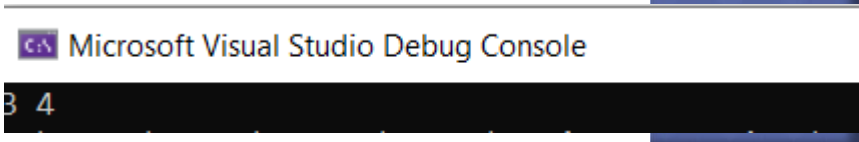


ЗАПИС (СТРУКТУРА)

- Извикване на член-функции
  - Работят с член-данните на класа
  - Извикват се с обект на класа

Пример:

```
struct Point2D {
  double x;
  double y;
  void print();
};


void Point2D::print() {
  std::cout << x << " " << y; // може да променим стойностите
}

int main() {
    Point2D myFirstPoint = { 3, 4 };
    myFirstPoint.print();
    return 0;
}
```



ЗАПИС (СТРУКТУРА)

- Извикване на член-функции
  - Работят с член-данните на класа
  - Извикват се с обект на класа

Пример:

```
struct Point2D {
  double x;
  double y;
  void print();
  void modifyX();
};
// …
void Point2D::modifyX() {
  x = 5; // може да променим стойностите
}

int main() {
    Point2D myFirstPoint = { 3, 4 };
    myFirstPoint.modifyX();
    myFirstPoint.print();
    return 0;
}
```

Запис (структура)

- Константни член-функции
  - Не променят член-данните на структурата.
  - Това се указва чрез записването на запазената дума const в декларацията и в края на заглавието в дефиницията им.
  - Извикват само const функции.

Запис (структура)

- Константни член-функции
  - Кои функции трябва да декларираме като константни?

Пример:

```
struct Point2D {
  double x;
  double y;
  void print() const;
};

void Point2D::print() const {
  std::cout << x << " " << y;
}
```

Запис (структура)

- Обекти – нека създадем повече екземпляри от структурата

Пример:

```
struct Point2D {
  double x;
  double y;
  void print() const;
};
// …
Point2D p1;
Point2D p2, p3;
// …
```

Запис (структура)

- Обекти - екземпляри на класа/структурата.

Пример:

```
struct Point2D {
  double x;
  double y;
  void print() const;
};
// …
Point2D p1;
Point2D p2, p3;
// …


// Какво сме постигнали?
```

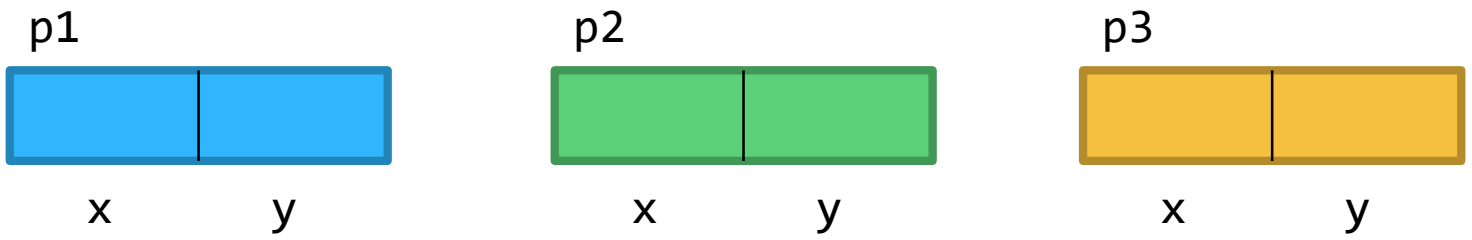
Запис (структура)

- Памет – заделя се при създаването на обект

Пример:

```
struct Point2D {
  double x;
  double y;
  void print() const;
};

Point2D p1;
Point2D p2, p3;
```

- Всеки обект разполага със собствени член-данни.

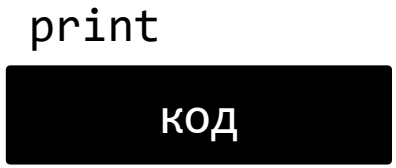


Запис (структура)

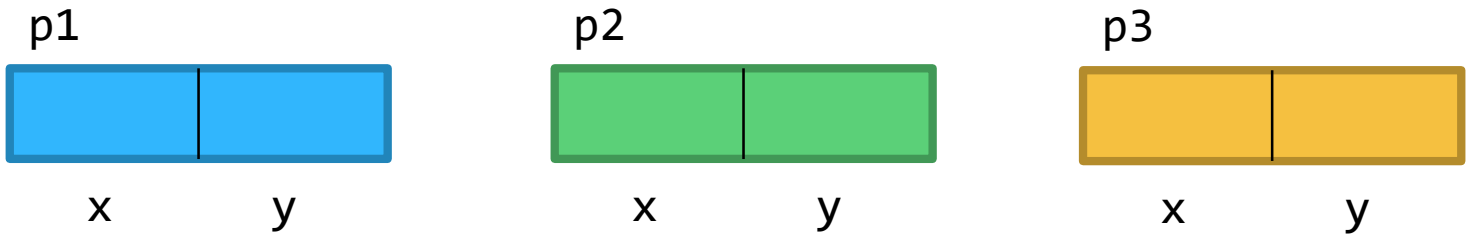
- Памет

Пример:

```
struct Point2D {
  double x;
  double y;
  void print() const;
};

Point2D p1;
Point2D p2, p3;
```



- Кодът на методите на класа НЕ се копира във всеки обект, а се намира само на едно място в паметта.

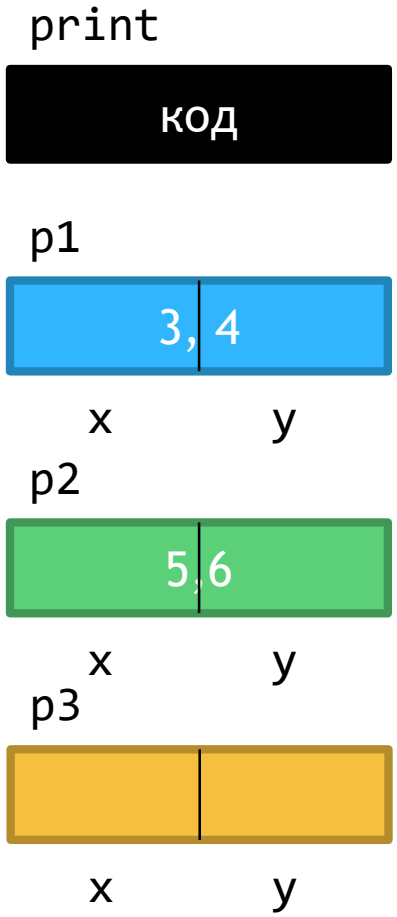


Запис (структура)

- Памет

Пример:

```
struct Point2D {
  double x;
  double y;
  void print() const;
};
void Point2D::print() const {
  std::cout << x << " " << y;
}
//…
Point2D p1 = {3.5, 4.2};
Point2D p2 = {5, 6}, p3;

p1.print();
p2.print();
```



По какъв начин член-данните на една структура “разбират” за кой обект на тази структура са били извикани?

Указател this

Работа на компилатора

- Преобразува всяка член-функция на дадена структура в обикновена функция с уникално име и един допълнителен параметър – указател this.

```
// Неконстантни функции
void Point2D::print() {
  std::cout << x << " " << y;
}
```

→

```
void Point2D::print(Point2D * this) {
  std::cout << this->x << " " << this->y;
}
```

Указател this

Работа на компилатора

- Преобразуване на константна член-функция

```
void Point2D::print() const {
  std::cout << x << " " << y;
}
```

→

```
// Стойностите не могат да се променят през указателя
void Point2D::print(Point2D const * this) const {
  std::cout << this->x << " " << this->y;
}
```

* *Припомнете си разликата между*
`Point2D const * this && Point2D * const this`

Указател this

Работа на компилатора

- Всяко обръщение към член-функция се транслира в съответствие първата част.

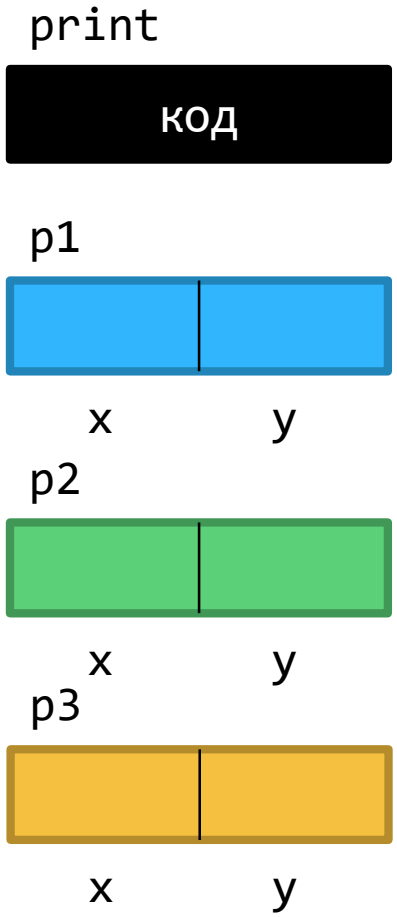
```
Point2D p1;
Point2D p2, p3;

p1.print(); → print(&p1);
p2.print(); → print(&p2);
```

Запис (структура)

- Памет

Пример:

```
struct Point2D {
  double x;
  double y;
  void print() const;
};
// Член-функция
void Point2D::print() const {
  std::cout << x << " " << y;
}
```



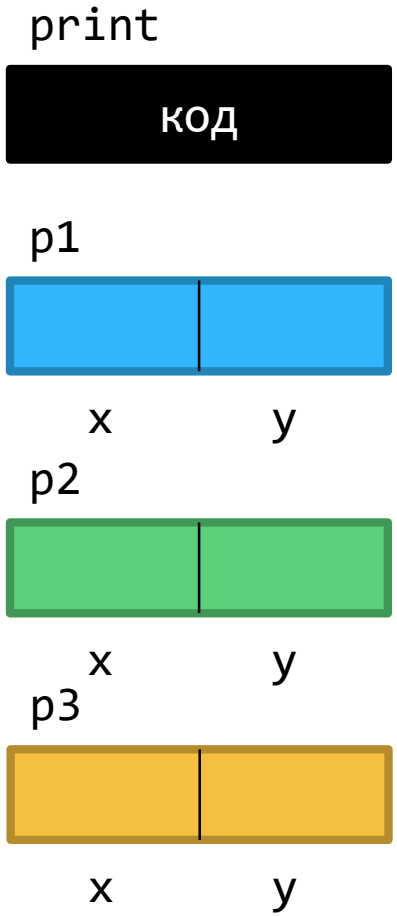
- Външни функции
  - Не са член-функции на структурата
  - Как създаваме еквивалентна външна функция, която работи с член-данните на структурата

Запис (структура)

- Памет

Пример:

```
struct Point2D {
  double x;
  double y;
  void print() const;
};
// Член-функция
void Point2D::print() const {
  std::cout << x << " " << y;
}
```



- Външни функции

```
// Няма неявен this параметър, нужен ни е обект
void print(Point2D const & p1) {
  std::cout << p1.x << " " << p1.y;
}
```

Запис (структура)

- Други специфики за външните функции

```
struct Point2D {
  double x;
  double y;
};
```

Какъв е типът? Какво ще се случи?

```
bool isInFirstQuadrant(Point2D p) {
  return p.x > 0 && p.y > 0;
}
```

Запис (структура)

- Други специфики за външните функции

```
struct Point2D {
  double x;
  double y;
};
```

Какъв е типът? Какво ще се случи?

```
Point2D getWesternPoint (Point2D p1, Point2D p2) {
  if (p1.x < p2.x) {
    return p1;
  }

  return p2;
}
```

ВРЕМЕ ЗА ВАШИТЕ ВЪПРОСИ

?